DOCUMENT RESUME

ED 304 094 IR 013 630

AUTHOR Anbar, Michael
TITLE CASIP--A Novel Authoring Tool for Open Ended Natural Language CAI.
PUB DATE Nov 86
NOTE 7p.; Paper presented at the International Conference of the Association for the Development of Computer-based Information Systems (28th, Washington, DC, November 10-13, 1986).
PUB TYPE Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Authoring Aids (Programing); *Computer Assisted Instruction; Individualized Instruction; *Instructional Design; Man Machine Systems; *Programing; *Programing Languages
IDENTIFIERS *Natural Language; *Open Ended Questions

ABSTRACT

Computer-assisted instruction that uses open ended questions and calls for answers in natural language is the preferred method of instruction in many training situations. A novel authoring tool has been developed to meet the specific needs of open ended computerized instruction. This program, named CASIP, is essentially a single purpose interpreter which uses instructions contained in an ASCII file to interpret natural language in a prescribed context and to respond accordingly. This and other ASCII files, which contain the questions and the responses used during a learning session, are completely loaded into memory at the beginning of the session. The use of CASIP involves no preprocessing of the ASCII input files. To author computer-assisted instruction with CASIP requires no programming knowledge, and it can be done with any word processor or line editor. CASIP can be used to generate sophisticated open ended instructional sessions on any subject in just a few days. (4 references) (Author/EW)

**********
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
**********

ED304094



# CASIP - A novel authoring tool for open ended natural language CAI

Michael Anbar PhD

Department of Biophysical Sciences, Sch. of Medicine, SUNY, Buffalo

ABSTRACT

CAI that uses open-ended questions, and calls for answers in natural language, is the preferred method of instruction in many training situations. A novel authoring tool has been developed to meet the specific needs of open-ended computerized instruction. This program, named CASIP, is essentially a single-purpose interpreter (written in C) which uses instructions contained in an ASCII file to interpret natural language in a prescribed context and to respond accordingly. This and other ASCII files, which contain the questions and the responses used during a learning session, are completely loaded into memory at the beginning of the session. The use of CASIP involves no preprocessing of the ASCII input files. To author CAI with CASIP requires no programming knowledge and it can be done with any word processor or line editor. CASIP can be used to generate in a few days sophisticated open-ended instructional sessions on any subject.

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
X This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Michael Anbar
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

## Open-ended versus Multiple Choice CAI

The popular multiple choice question (MCQ) approach in instruction has serious limitations. Its major drawbacks are: (1) It is the instructor and not the student who formulates the alternative answers to a given problem. (2) It assumes that a finite small number of plausible answers (including credible detractors) constitute essentially all the reasonable alternative answers to a given problem. (3) It implies unambiguous problems. Except in mathematics or in trivial recall of facts, there may be many reasonable answers to a question, some of which may not have occured to the instructor. Still, the student is expected to select the "correct" one(s). (4) It does not allow the student to explain the reasoning behind a certain choice; a student may come up with the "right" answer for a wrong reason, or with a "wrong" answer for a quite valid reason. Students who think creatively may be, therefore, at a disadvantage in a MCQ environment.

These limitations are pertinent to most instructional situations, with the possible exception of trivial memorization and recall drilling. This is true of high school and undergraduate college teaching as it is in the teaching of medicine, law, economics or any other subject that requires conceptual problem solving. Further, in computerized MCQ students can get immediate feedback to their answers. However, such feedback just confirms or contradicts the answer and not critically analyse the rationale for

IR013630

the given choice. This is generally not done because it is very difficult, if not impossible, to apply MCQ's to examine the reasoning of the student without introducing biases - suggesting reasoning the student may never have thought of.

The alternative to MCQ is cue-free open-ended questions (OEQ). Here the students who are presented with a problem must come up with a plausible solution all on their own. Any teacher who has graded short assays or open-answer questions has realized how more informative they are about the student's knowledge and mode of reasoning, in comparison to MCQ. In fact, coming up with a set of alternative potential answers to a given problem is an experience that often is educationally more valuable and intellectually more rewarding than chosing the "right" alternative from among those answers. When used in an iterative manner OEQ are well suited for the age proven socratic interactive mode of instruction which is based on a teacher-student dialog that effectively explores and criticizes the reasoning of the student. Furthermore, the OEQ modality of CAI is a man-machine interactive experience where both parties use essentially the same language.

Programming of Open-ended CAI

The computer has proven to be highly efficient in handling MCQ, especially since it facilitates multiple stage conditional branching. However, to become a major component in advanced academic instruction, the computer must be capable of handling OEQ -it must be capable of understanding natural language. The handling of the whole universe of words and expressions in response to a question may seem freightening at first sight. However, based on first hand experience with this type of CAI for several years [Anbar, 1984,1985,1986a,1986b], I can state that the range of principal key-words (each with a number of, say, up to 10, logical synonymes) that may show up in a coherent answer following a well structured open ended question is rather limited (between 5 and 20). This relatively small number of string recognitions is well within the capacity of computers; any microcomputer for that manner. But it still means the precise recognition of over a hundred words or phrases (a phrase in this context is a coherent combination of two or more words) for each question, which is not trivial from the programming standpoint.

Using the classical programming approaches and high languages such as BASIC or Pascal it is far more difficult to program OEQ sessions than branched MCQ, and the time and effort required make it virtually impossible to generate instructional materials of acceptable quality in quantities that can might have an impact on the curriculum. Unfortunately we also cannot benefit from the use of commercially available "authoring languages". When applied to our specific goal of generating effective OEQ materials, non of the available software has mets our needs. For one, none of these packages can handle hundreds of recognizable words and tens of potential answers to each question and then branch accordingly into tens of branches, repeating the same diversity of response over and over again. Even if coded to do so by a tour de force, the response

times obtained we observed with the programs we tested were prohibitively long (tens of seconds per cycle!). Since these programs have not been designed to handle our type of problem, the results are not surprising.

Special Requirements of Open-end CAI

The only viable solution to our problem has remained the generation de novo of an authoring tool that will meet our specific needs. These needs can be generically expressed in the following five requirements:

A. Since the recognition of whole sentences or parts thereof is utterly impractical without parcing into words, we need a program that will allow the rapid (within less than 2 seconds) matching of each word in a sentence of a student's answer (averaging, say, 8 words) with any of a few hundreds of key words at a time. The program must also be able, within the same time frame, to branch to any desireble response after recognizing 1 to 6 different key-words in a sentence. The recognized words may appear in a prescribed sequence and at a specified separation from each other (adjacent two words, or two words within, say, four words of each other). It must also readily allow alternative treatment of an answer upon recognition of the presence of certain words (eg. "not" or "never") that might change the meaning of a sentence (this cannot be easily implemented in the commercial software tested).

B. The response to a recognized word or sequence of words or to the absence of a word or sequence of words in the student's input, may be a short sentence, with a follow-on question (that specifies its expected range of responses), or it may be a display of text or tabulated numeric data (ranging from several lines to several sceens), computer stationary graphics (binary stored pictures), in-situ generated graphics or computer animation, followed by the next question. The program must also recognize a repeated answer that might prompt the same response (to avoid logical loops). The program must allow to make the response conditional on answering a prescribed previous question (or number of questions); this requirement is essential when analysing the reasoning of a student. It must also provide a different response after the student has given a prescribed number of unexpected answers to a given question (to avoid the student's frustration). Finally, it must allow the incorporation of a preselected key word from the student's answer into the response or the follow on question; this gives the student the assurance that the computer really "understands".

C. The program must produce a permanent record of the student's answer (verbatim), of the question that evoked that answer and of the response the machine gave following the given answer. This is essential for the development of any OEQ material. Having a record of the unexpected answers of several students in trial runs is absolutely necessary to fine tune the program before its deployment to the whole class; the author alone cannot anticipate all possible answers to each question.

D. The program must have the capability to display randomly selected questions or responses from a specified series, in situations where virtually the same response is called for (eg. responding to an unexpected answer or to a repeated answer), to make the machine look less predictable and, therefore, less mechanistic. This feature is especially important when the same material is to be used several times by the same student.

E. The input to the program - questions, responses, data, as well as the correlation between key-words recognized in the student's answers and the corresponding responses, must be presented in a simple format, so that an author without any programming experience should be able to write lessons with minimal difficulty. The author should have to do minimal manipulations when incorporating the authored inputs into a working program (preprocessing or compilation should be avoided). This is especially critical during development of a complex OEQ session and especially when additions and corrections are made repeatedly following trials with students.

Considering this combination of requirements it is clear why it was not met by any of the available commercial authoring languages; the developers of the latter did not have these objectives in mind. What we have done is to use the C language, which combines high flexibility with speed of processing, to develop a programming tool that meets all these goals and more.

CASIP - an Interpreter for Open-ended CAI

We have called our OEQ program CASIP (Computer Assisted Socratic Instruction Program) since it facilitates computer-mediated socratic teaching. CASIP has been written in the C language in a fairly compact manner (<40K compiled, with a source code of about 30K). It runs on a 192K IBM PC (with DOS 2.0 or 2.1, or 256K with DOS 3.0 or 3.1) and is expected to be transportable to any MSDOS 8086/8088 system. It can run on a mono monitor (without graphics), on a color monitor, or on a dual mono/color system (ideally). Since the graphics are external to the core program, any graphics (or music, if desired) compiled in an .EXE file, or binary coded .PIC file screens (16K), can be displayed as the response.

All the author's inputs - the questions, lists of recognizing words, responses and the correlation table, which provides the run-time correspondence between the recognized key-words and the responses, are simple ASCII lists (they can be produced on any word processor or line editor). All these files are completely loaded into memory before the program starts. This facilitates a high speed of processing of the special algorithms that minimize the time of answer analysis. Depending on the particular lesson, these ASCII inputs may amount to as much as 100K combined, but as little as 20K input (about 5 typed pages) can produce a very effective instructional unit.

The program reads external text files, it rapidly reads binary graphics files and executes external graphics .EXE files (in

compiled BASIC, Pascal or C). It should be emphasized that the use of CASIP requires no preprocessing or compilation. In fact, this program acts as a single-purpose interpreter which immediately runs the loaded ASCII files. It records the student's answers fully, keeps a record of the preceeding question and the follow-on reponse, and keeps a record of the time (with a precision of seconds) each answer was given; this helps to identify difficult questions or casual, not sufficiently thoughtful answers. The record includes also the number of times the student gave an unexpected answer and the score; each response or question, if you wish, may have a score assigned by the author. The student's record is eventually written into an individualized hidden (optional, activated by external control) file.

The program can also be instructed to terminate at a predetermined time between 10 and 90 minutes (set at 10 minute intervals), or after a predetermined number of question-answer cycles. It can be also terminated as one of the responses following a certain answer - termination on accomplishment of a prescribed learning task. A special instruction traps control-C termination and warm booting to avoid unwarranted termination by a frustrated user. The instructor can, however, terminate the session at any time using a special secret (known only to the instructor) key-word. Another instruction calls for a continuous clock display on the screen.

The options of word recognition include the recognition of a single or a sequence of words (usually 2 to 6 words); the words in a sequence do not have to be adjacent to each other and the author may specify how many other words can be accepted between each word pair in a sequence. The author has also the option to allow two words to be recognized irrespecive of the order they appear in the answer. The author can also specify one response to a combination of two words (in sequence or, optionally, out of sequence), another response if only the first word was recognized, and yet another response if the second word alone was recognized. As mentioned above, there is the option to modify the response even when the desireable sequence of key- words was recognized, if a certain other word (eg. "no" or "never") were recognized in the answer. Anther option is to display automatically a certain response after the student answered satisfacorily a certain number (say, four) out of the possible answers (there may be, say, 12 different answers to a given question the author anticipated) to a given question. Another automatic response may be given if the student gave in one sequence a certain (say, three) unrecognized (wrong or unexpected) answers.

## CASIP in Use

CASIP gives the author all the necessary flexibility to recognize the contents of a student's answer at any desireable level of precision. Obviously, the more restrictive the prescription of the expected answer the more answers must be prescribed to a given question or the greater are the chances to receive an unexpected answer. In other words, the author must learn

to compromize between the chance of accepting a poorly matching answer (a false positive, in the medical lingo) and unjustifiably rejecting an acceptable answer (a false negative). Trial runs with students is the only way to minimize authoring errors of either kind. The ease of obtaining a record of the students' answers and the ease of correcting the input files make this iterative process very easy. It takes just an hour or two to make substantial revisions in the files following a productive trial run.

The authoring process in generating a CASIP session has essentially nothing to do with computer programming. What the author has to do is to carry out an imaginary dialog with a student and type the questions, the keyword-words to be recognized in the "student's" answers, and the responses, in a conventional word processor file. The rest includes separating the three types of inputs into three different files (by block movement and transfer), and the construction of the correlation table that instructs CASIP which response will follow a given set of recognized key-words, which question will follow that response and what group of key-words to look for in the answer to that question. Because of these minimal requirements, the authoring of a fairly sophisticated session takes only two to three days. Including the repeated testing on students, a session can be ready for class use in a week or two.

We have tested CASIP on a variety of subjects ranging from emergency medicine to social studies at the high school level, and the results are highly encouraging. We have thus demonstrated that effective open ended CAI, which uses unrestricted natural language as its input and can be applied to any academic discipline at any level of education, can be produced with an unexpectedly low level of effort.

REFERENCES

[1] Anbar M., (1984) Experiences with CAI in Medical Education. in Capitol-izing on Computers in Education, Proc. 1984 AEDS Convention, C.D. Martin and R.S. Heller, Editors, Computer Science Press, Rockville, 1984, pp 188-190.

[2] Anbar R.D., Anbar M. and J.M. Hassett Jr, (1985) Emergency Medicine and Intensive Care Simulators. Proc. 9th Annual Symposium on Computer Applications in Medical Care (SCAMC), Ackerman M., Editor, IEEE Computer Soc, pp 547-550.

[3] Anbar M., (1986a) Open-ended Computer Based Medical Instruction - Avoiding the Pitfalls of Multiple Choice Behavior. Resident & Staff Physician, August 1986, in print.

[4] Anbar M., (1986b) The Socratic Teacher - Computer Assisted Self Instruction. Ch 9 in: Computers in Medicine, M. Anbar, Editor, Computer Science Press, Rockville, 1986.